

EECS 22: Advanced C Programming

Lecture 17

Rainer Dömer

doemer@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

Lecture 17: Overview

- Course Administration
 - Reminder: Final course evaluation
- Command Line Arguments
 - Parameters to function `main`
- File Processing
 - Introduction
 - Standard input and output
 - I/O streams, file streams
 - Standard library functions in `stdio.h`
- Command Line Tool: `students`
 - Tool to sort student records stored as text files
 - `students.c`
 - `Makefile`

Course Administration

- Final Course Evaluation
 - Open until end of 10th week (Sunday night)
 - Nov. 20, 2012, through Dec. 9, 2012, 11:45pm
 - Online via EEE Evaluation application
- Mandatory Evaluation of Course and Instructor
 - Voluntary
 - Anonymous
 - Very valuable
- Your feedback is appreciated!

Command Line Arguments

- Operating System
 - ... passes optional arguments to programs started from the command line
 - `echo Hello World`
 - `gcc HelloWorld.c -ansi -Wall -o HelloWorld`
- C Programs
 - ... can opt to ignore the arguments
 - `int main(void);`
 - function `main` takes no arguments
 - ... or, can receive arguments as parameters to main function
 - `int main(int argc, char *argv[]);`
 - function `main` takes two arguments:
 - `argc` specifies the number of arguments provided
 - `argv` is an array of pointers to arguments of type string

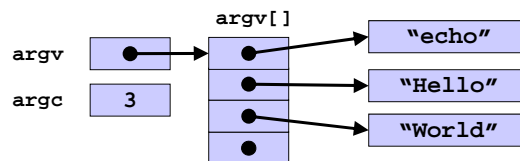
Command Line Arguments

- Parameters to Function `main`

- `int main(int argc, char *argv[]);`
 - `argc` specifies the number of arguments provided *including* the program name (!)
 - `argv` is an array of pointers to arguments of type string
 - `argv[0]` is the program name (command)
 - `argv[1...argc-1]` are the arguments specified on the command line
 - `argv[argc]` is a NULL pointer

– Example:

```
% echo Hello World
Hello World
%
```



File Processing

- Introduction

- Variables store data only during program run time
 - At program completion, data in main memory is lost
- *Persistent data* is stored even after a program exits
- Persistent data is stored as *files* in a *file system*
 - ... on the hard disk,
 - ... on a removable disk (CD, DVD, memory stick, ...),
 - ... in the network,
 - ... etc.
- Input and output from and to files is organized as data streams: *I/O streams*

File Processing

- I/O Streams
 - Standard I/O streams (opened by the system)
 - `stdin` standard input stream (i.e. `scanf()`)
 - `stdout` standard output stream (i.e. `printf()`)
 - `stderr` standard error stream (i.e. `perror()`)
 - File I/O streams (explicitly opened by a program)
 - Open a file `fopen()`
 - Write data to a file `fprintf()`, `fputs()`, etc.
 - Read data from a file `fscanf()`, `fgets()`, etc.
 - Close a file `fclose()`
 - In C, I/O streaming functions are ...
 - ... declared in header file `stdio.h`
 - ... provided by the standard C library

EECS22: Advanced C Programming, Lecture 17

(c) 2012 R. Doemer

7

Standard I/O Functions

- Functions declared in `stdio.h` (part 1/4)
 - `int printf(const char *fmt, ...);`
 - `int scanf(const char *fmt, ...);`
 - formatted output/input to/from stream `stdin/stdout`
 - `int sprintf(char *s, const char *fmt, ...);`
 - `int sscanf(const char *s, const char *fmt, ...);`
 - formatted output/input to/from a string `s`
 - `int getchar(void);`
 - `int putchar(int c);`
 - input/output of a single character to/from stream `stdin/stdout`
 - `char *gets(char *s);`
 - `int puts(const char *s);`
 - input/output of strings to/from stream `stdin/stdout`

EECS22: Advanced C Programming, Lecture 17

(c) 2012 R. Doemer

8

Standard I/O Functions

- Functions declared in `stdio.h` (part 2/4)
 - `typedef __FILE FILE;`
 - opaque type for a file handle
 - `FILE *fopen(const char *n, const char *m);`
 - open file named `n` for input ("`r`"), output ("`w`"), or append ("`a`")
 - returns a file handle, or `NULL` in case of an error
 - `int fclose(FILE *f);`
 - closes an open file handle
 - `int fprintf(FILE *f, const char *fmt, ...);`
 - `int fscanf(FILE *f, const char *fmt, ...);`
 - `int fgetc(FILE *f);`
 - `char *fgets(char *s, int n, FILE *f);`
 - `int fputc(int c, FILE *f);`
 - `int fputs(const char *s, FILE *f);`
 - input/output functions from/to stream `f`
 - `int fflush(FILE *f);`
 - flushes any unwritten data from a buffer into the file

Standard I/O Functions

- Functions declared in `stdio.h` (part 3/4)
 - `typedef unsigned int size_t;`
 - type for size of a block of memory (number of bytes)
 - `size_t fread(void *p, size_t s, size_t n, FILE *f);`
 - binary input to memory location `p` for `n` times `s` bytes from file `f`
 - `size_t fwrite(const void *p, size_t s, size_t n, FILE *f);`
 - binary output from memory location `p` for `n` times `s` bytes to file `f`
 - `int fseek(FILE *f, long pos, int w);`
 - move to position `pos` in file `f` (from beginning/current pos/end)
 - `long ftell(FILE *f);`
 - return the current position in file `f` (from beginning)
 - `void rewind(FILE *f);`
 - move to beginning of file `f`
 - `int feof(FILE *f);`
 - check if end of file `f` is reached

Standard I/O Functions

- Functions declared in `stdio.h` (part 4/4)
 - `int ferror(FILE *f);`
 - returns the current error status for file `f`
 - `void perror(const char *prg);`
 - print current error for program `prg` to stream `stderr`
 - `int remove(const char *filename);`
 - delete file `filename`
 - `int rename(const char *old, const char *new);`
 - rename file `old` to new name `new`

Command Line Tool: `students`

- In order to practice...
 - Command Line Arguments
 - File Processing
 - Standard I/O streams and functions
- ..., let's build a simple command line tool that sorts student records stored in text files
- Example:

```
% students -h
Usage: students [option...]
Option:      Function:
-h           print this usage info
-v           be verbose (default: be quiet)
-i <file>    specify input filename (default: stdin)
-o <file>    specify output filename (default: stdout)
-id          sort student records by ID
-name        sort student records by name (default)
-grade      sort student records by grade
%
```

Command Line Tool: **students**

- Example:
 - Text file with student data:

1001	Jane Doe	A
1002	John Doe	C
1000	New Kid	F
1003	Jane Doe	B
1009	Z End	A
1008	Alice A	A
1007	Bob B	B
1006	Carl C	C
1005	Dave D	D
1004	Eve E	F

- Example run:

```

% vi ListOfStudents.txt
% students -i ListOfStudents.txt -name
1008 Alice A           A
1007 Bob B             B
1006 Carl C            C
1005 Dave D            D
1004 Eve E             F
1001 Jane Doe          A
1003 Jane Doe          B
1002 John Doe          C
1000 New Kid           F
1009 Z End             A
...
    
```

EECS22: Advanced C Programming, Lecture 17
(c) 2012 R. Doemer
13

Command Line Tool: **students**

- **Reuse** the Sorted Double-Linked List
 - Example: List of Student Records sorted by ID, Name, or Grade

Length	42			
First	●			
Last	●			

List Next Prev Student	●	●	●	●
	●	●	●	●
	●	●	●	●
	●	●	●	●

ID	1001	1003	1002	...	1042
Name	"Jane Doe"	"Jim Doe"	"John Doe"	...	"Z End"
Grade	'A'	'B'	'C'	...	'A'

EECS22: Advanced C Programming, Lecture 17
(c) 2012 R. Doemer
14

Command Line Tool: students

- Example `students.c` (part 1/9)

```

/* students.c: command-line tool to sort student records */
/* author: Rainer Doemer */
/* 11/21/11 RD initial version */

#include "StudentSort.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <assert.h>

void PrintUsage(char *ProgramName)
{
    fprintf(stderr, "Usage: %s [option...]\n",
            "Option:      Function:\n",
            "-h           print this usage info\n",
            "-v           be verbose (default: be quiet)\n",
            "-i <file>    specify input filename (default: stdin)\n",
            "-o <file>    specify output filename (default: stdout)\n",
            "-id          sort student records by ID\n",
            "-name        sort student records by name (default)\n",
            "-grade       sort student records by grade\n",
            ProgramName);
}
...

```

EECS22: Advanced C Programming, Lecture 17

(c) 2012 R. Doemer

15

Command Line Tool: students

- Example `students.c` (part 2/9)

```

...
SLIST *ReadStudentFile(char *InFileName, int Verbose,
                      char *ProgramName)
{
    SLIST *l = NULL;
    FILE *InFile = stdin;
    int n, ID;
    char FirstName[SLEN], LastName[SLEN], FullName[SLEN*2+1];
    char Grade;
    if (Verbose)
        { fprintf(stderr, "Reading from file %s...\n",
                (InFileName ? InFileName : "stdin"));
        }
    if (InFileName)
        { InFile = fopen(InFileName, "r");
          if (!InFile)
            { fprintf(stderr, "%s: Can't open file \"%s\" to read!\n",
                    ProgramName, InFileName);
              perror(ProgramName);
              exit(10);
            }
        }
    ...
}
...

```

EECS22: Advanced C Programming, Lecture 17

(c) 2012 R. Doemer

16

Command Line Tool: **students**

- Example **students.c** (part 3/9)

```

...l = NewStudentList();
while(1)
{ assert(SLEN >= 20);
  n = fscanf(InFile, " %d %20s %20s %c",
            &ID, FirstName, LastName, &Grade);

  if (n == EOF)
  { break; }
  if (n != 4)
  { fprintf(stderr, "%s: Read incomplete record!\n",
            ProgramName);
    exit(10); }
  sprintf(FullName, "%s %s", FirstName, LastName);
  if (Verbose)
  { fprintf(stderr, "Read student ID %d, name \"%s\", "
                "grade %c...\n", ID, FullName, Grade); }
  if (FindStudentID(l, ID))
  { fprintf(stderr, "%s: Found multiple IDs %d!\n",
            ProgramName, ID);
    exit(10); }
  InsertStudent(l, NewStudent(ID, FullName, Grade),
                CompareStudentID);
}...

```

EECS22: Advanced C Programming, Lecture 17

(c) 2012 R. Doemer

17

Command Line Tool: **students**

- Example **students.c** (part 4/9)

```

...

if (InFile != stdin)
{ fclose(InFile);
  InFile = NULL;
}
if (Verbose)
{ fprintf(stderr, "Successfully read %d student records!\n",
            l->Length);
}
return(l);
} /* end of ReadStudentFile */

...

```

EECS22: Advanced C Programming, Lecture 17

(c) 2012 R. Doemer

18

Command Line Tool: students

- Example `students.c` (part 5/9)

```

...
void WriteStudentFile(char *OutFileName, int Verbose,
                     char *ProgramName, SLIST *l)
{
    FILE *OutFile = stdout;
    SENTRY *e;

    if (Verbose)
        { fprintf(stderr, "Writing to file %s...\n",
                  (OutFileName ? OutFileName : "stdout"));
        }
    if (OutFileName)
        { OutFile = fopen(OutFileName, "w");
        if (!OutFile)
            { fprintf(stderr, "%s: Can't open file \"%s\"
                        " to write!\n",
                        ProgramName, OutFileName);
              perror(ProgramName);
              exit(10);
            }
        }
}
...

```

EECS22: Advanced C Programming, Lecture 17

(c) 2012 R. Doemer

19

Command Line Tool: students

- Example `students.c` (part 6/9)

```

...
    e = l->First;
    while(e)
        { assert(e->Student);
          fprintf(OutFile, "%4d %-45s %c\n",
                  e->Student->ID, e->Student->Name,
                  e->Student->Grade);

          e = e->Next;
        }
    if (OutFile != stdout)
        { fclose(OutFile);
          OutFile = NULL;
        }
    if (Verbose)
        { fprintf(stderr, "Successfully wrote %d records!\n",
                    l->Length);
        }
} /* end of WriteStudentFile */
...

```

EECS22: Advanced C Programming, Lecture 17

(c) 2012 R. Doemer

20

Command Line Tool: **students**

- Example **students.c** (part 7/9)

```

...
int main(int argc, char *argv[])
{
    int i, Verbose = 0;
    char *InFileName = NULL, OutFileName = NULL;
    CMP_F *CompareFct = CompareStudentName;
    SLIST *l = NULL;
    for(i=1; i<argc; i++)
    {
        if (0 == strcmp(argv[i], "-h"))
        {
            PrintUsage(argv[0]);
            exit(0);
        }
        else if (0 == strcmp(argv[i], "-v"))
        {
            Verbose = 1;
        }
        else if (0 == strcmp(argv[i], "-i"))
        {
            i++;
            if (i < argc)
            {
                InFileName = argv[i];
            }
            else
            {
                fputs("Missing argument for -i!\n", stderr);
                PrintUsage(argv[0]);
                exit(10);
            }
        }
    }
    ...

```

EECS22: Advanced C Programming, Lecture 17

(c) 2012 R. Doemer

21

Command Line Tool: **students**

- Example **students.c** (part 8/9)

```

...
        else if (0 == strcmp(argv[i], "-o"))
        {
            i++;
            if (i < argc)
            {
                OutFileName = argv[i];
            }
            else
            {
                fputs("Missing argument for -o!\n", stderr);
                PrintUsage(argv[0]);
                exit(10);
            }
        }
        else if (0 == strcmp(argv[i], "-id"))
        {
            CompareFct = CompareStudentID;
        }
        else if (0 == strcmp(argv[i], "-name"))
        {
            CompareFct = CompareStudentName;
        }
        else if (0 == strcmp(argv[i], "-grade"))
        {
            CompareFct = CompareStudentGrade;
        }
        else
        {
            fprintf(stderr, "Unknown option %s!\n", argv[i]);
            PrintUsage(argv[0]);
            exit(10);
        }
    }
    ...

```

EECS22: Advanced C Programming, Lecture 17

(c) 2012 R. Doemer

22

Command Line Tool: **students**

- Example **students.c** (part 9/9)

```

...

    l = ReadStudentFile(InFileName, Verbose, argv[0]);

    SortStudentList(l, CompareFct);

    WriteStudentFile(OutFileName, Verbose, argv[0], l);

    DeleteStudentList(l);
    l = NULL;
    return 0;
} /* end of main */

/* EOF */

```

Command Line Tool: **students**

- Example **Makefile** (part 1/3)

```

# Makefile: Student Records

# macro definitions
CC = gcc
DEBUG = -g
#DEBUG = -O2
CFLAGS = -Wall -ansi $(DEBUG) -c
LFLAGS = -Wall -ansi $(DEBUG)
MAIN = -DMAIN

# dummy targets
all: Student StudentList StudentSort students

test: all
    valgrind Student
    valgrind StudentList
    valgrind StudentSort
    valgrind students

clean:
    rm -f *.o
    rm -f Student StudentList StudentSort students

...

```

Command Line Tool: **students**

- Example **Makefile** (part 2/3)

```

...
# compilation rules
Student.o: Student.c Student.h
    $(CC) $(CFLAGS) Student.c -o Student.o

Student: Student.c Student.h
    $(CC) $(MAIN) $(LFLAGS) Student.c -o Student

StudentList.o: StudentList.c StudentList.h Student.h
    $(CC) $(CFLAGS) StudentList.c -o StudentList.o

StudentList: StudentList.c StudentList.h Student.h Student.o
    $(CC) $(MAIN) $(LFLAGS) StudentList.c Student.o \
        -o StudentList

StudentSort.o: StudentSort.c StudentSort.h StudentList.h Student.h
    $(CC) $(CFLAGS) StudentSort.c -o StudentSort.o

StudentSort: StudentSort.c StudentSort.h StudentList.h Student.h \
    StudentList.o Student.o
    $(CC) $(MAIN) $(LFLAGS) StudentSort.c StudentList.o \
        Student.o -o StudentSort
...

```

EECS22: Advanced C Programming, Lecture 17

(c) 2012 R. Doemer

25

Command Line Tool: **students**

- Example **Makefile** (part 3/3)

```

...

students: students.c StudentSort.h StudentList.h Student.h \
    StudentSort.o StudentList.o Student.o
    $(CC) $(MAIN) $(LFLAGS) students.c -o students \
        StudentSort.o StudentList.o Student.o

# EOF

```

EECS22: Advanced C Programming, Lecture 17

(c) 2012 R. Doemer

26

Command Line Tool: **students**

- Example Session

```
% vi students.c
% vi Makefile
% make students
gcc -Wall -ansi -g -c StudentSort.c -o StudentSort.o
gcc -Wall -ansi -g -c StudentList.c -o StudentList.o
gcc -Wall -ansi -g -c Student.c -o Student.o
gcc -DMAIN -Wall -ansi -g students.c -o students \
    StudentSort.o StudentList.o Student.o

% students -h
Usage: students [option...]
Option:      Function:
-h           print this usage info
-v           be verbose (default: be quiet)
-i <file>    specify input filename (default: stdin)
-o <file>    specify output filename (default: stdout)
-id          sort student records by ID
-name        sort student records by name (default)
-grade      sort student records by grade
...
```

Command Line Tool: **students**

- Example Session

```
...
% vi ListOfStudents.txt
% students -i ListOfStudents.txt -o List2.txt -v
Reading from file ListOfStudents.txt...
Read student ID 1001, name "Jane Doe", grade A...
Read student ID 1002, name "John Doe", grade C...
Read student ID 1000, name "New Kid", grade F...
Read student ID 1003, name "Jane Doe", grade B...
Read student ID 1009, name "Z End", grade A...
Read student ID 1008, name "Alice A", grade A...
Read student ID 1007, name "Bob B", grade B...
Read student ID 1006, name "Carl C", grade C...
Read student ID 1005, name "Dave D", grade D...
Read student ID 1004, name "Eve E", grade F...
Successfully read 10 student records!
Writing to file List2.txt...
Successfully wrote 10 student records!
...
```

Command Line Tool: `students`

- Example Session

```
...
% students -i ListOfStudents.txt -name
1008 Alice A           A
1007 Bob B             B
1006 Carl C            C
1005 Dave D            D
1004 Eve E             F
1001 Jane Doe          A
1003 Jane Doe          B
1002 John Doe          C
1000 New Kid           F
1009 Z End             A
...
```

Command Line Tool: `students`

- Example Session

```
...
% students -i ListOfStudents.txt -id
1000 New Kid           F
1001 Jane Doe          A
1002 John Doe          C
1003 Jane Doe          B
1004 Eve E             F
1005 Dave D            D
1006 Carl C            C
1007 Bob B             B
1008 Alice A           A
1009 Z End             A
...
```

Command Line Tool: `students`

- Example Session

```
...
% students -i ListOfStudents.txt -grade
1001 Jane Doe A
1008 Alice A A
1009 Z End A
1003 Jane Doe B
1007 Bob B B
1002 John Doe C
1006 Carl C C
1005 Dave D D
1000 New Kid F
1004 Eve E F
%
```